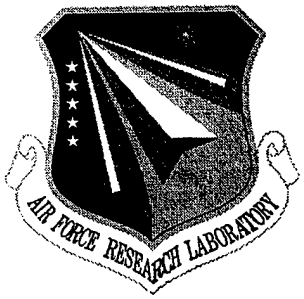
AFRL-IF-RS-TR-1999-235
Final Technical Report
October 1999



MULTIPROCESSOR STATION (MIPS)

Nichols Research Corporation

Jaye Bass

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1999-235 has been reviewed and is approved for publication.

APPROVED:

SCOTT F. ADAMS
Project Engineer

FOR THE DIRECTOR:

JOHN V. MCNAMARA, Technical Advisor
Information & Intelligence Exploitation Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFEC, 32 Brooks Road, Rome, NY 13441-4114. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Oct 99	Final Apr 96 - Jan 99

4. TITLE AND SUBTITLE

MULTIPROCESSOR STATION (MIPS)

5. FUNDING NUMBERS

C - F30602-96-C-0083
PE - 63260F
PR - 3480
TA - PD
WU - 49

6. AUTHOR(S)

Jaye Bass

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Nichols Research Corporation
4040 S. Memorial Parkway
PO Box 400002
Huntsville, AL 35815-1502

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFEC
32 Brooks Road
Rome, NY 13441-4114

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-1999-235

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Scott F. Adams, IFEC, 315-330-1430

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The objective of the MultiProcessor Station (MIPS) effort was to establish a software development environment to: maximize cross-platform portability of image and video processing software; maximize the utilization of available multiprocessors on intelligence workstations; and maximize the life-span and minimize the life-cycle cost of software. The end product is: an object-based visual programming environment in which image and video data and operations are represented by icons; an environment which permits mixing of legacy code and new development; a framework providing the capability to visually program new algorithms without writing any source code; tight integration of the constructor mode with the main window mode means that new algorithms can be developed and easily installed on any computer with MIPS software; the image cube object, which is used to store all imagery and video, makes it possible for all image processing algorithms to be applied to all types of data; versatility for exploiters of multi-spectral imagery, in that processing and display of image bands can be tightly controlled, easily modified, and packaged for widespread use; the environment demonstrates portability across platforms, software reusability, adaptability to various multiprocessor configurations, and extensibility to other application areas (i.e. signal processing).

14. SUBJECT TERMS

Image processing, parallel processing, video processing, visual programming, portable programming, spectral imaging

15. NUMBER OF PAGES

44

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

Table of Figures

1 Introduction

This report describes the activities and accomplishments of the "MultI-Processor Station (MIPS)" program performed by Nichols Research Corporation (NRC). The work was sponsored by Air Force Research Laboratory (AFRL), Multi-Sensor Exploitation Branch, Contract No. F30602-96-C-0083 from 30 April 1996 to 31 January 1999. Within the effort, the MIPS suite of re-usable object-oriented frameworks, the MIPS constructor and the MIPS Mainwindow were developed and delivered to AFRL.

The AFRL Program Manager was Mr. Scott Adams. The NRC Program Manager was Mr. Gary Grider. Mr. Jaye Bass served as technical lead. Mr. Brett Gossage developed the object server framework. Mr. John Spears developed the GUI's for both the Constructor and the Mainwindow. Ms. Marisa Wheelock developed the image processing algorithms. Mr. Chris Parker served as lead test engineer.

The primary objective for this effort is to improve the information flow to the warfighter by providing an automated image processing workstation which provides near real-time imagery data reduction. This was accomplished by developing the MIPS Constructor, an object oriented, parallel processing, visual programming image exploitation tool, and the MIPS Mainwindow, a window oriented image exploitation tool integrated with and upgradeable by the Constructor. **These tools provide the warfighter with an easy to use, field upgradeable, parallel processing solution to all source (RGB/grayscale, video, or multi/hyperspectral) image exploitation.** Additional objects of this effort included:

- Demonstration of the advantages of a multiprocessor workstation to exploit imagery.
- Construction of a prototype workstation using cost effective commercial off the shelf components integrating image and video analysis routines developed for the Image Exploitation 2000 (IE2000).
- Demonstrate the software reuse advantages of an object oriented software development environment.
- Demonstrate an innovative cost effective upgrade path for existing Air Force assets.

2 Overview

In response to the objectives stated on the previous page, NRC has developed 3 core products.

2.1 MIPS Frameworks

The first product is a set of re-usable object oriented frameworks:

- Image Processing (IP) Framework
- InterProcess Communication and Control (IPCC) Framework
- Image Formats (IF) Framework
- Parallel Image Processing (PIP) Framework
- Visual Imaging Environment (VIE) Framework

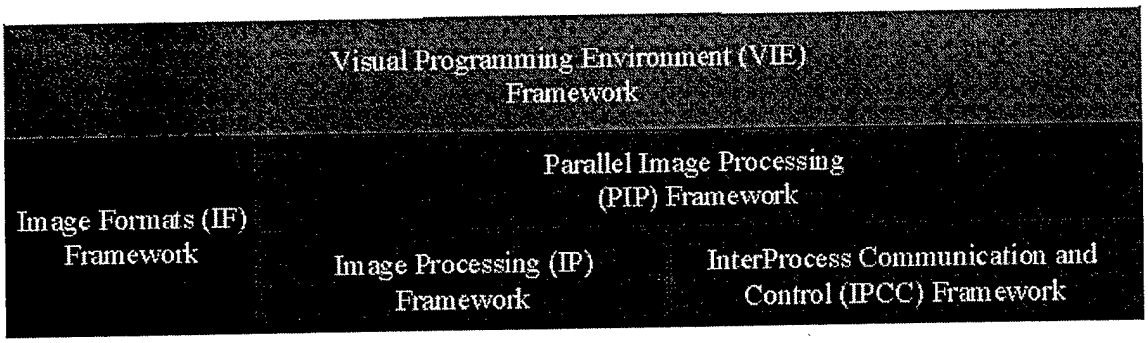


Figure 1. Illustrated above is a graphical representation of the MIPS frameworks.

Each of these frameworks, illustrated in figure 1, was carefully designed for re-use, portability, extensibility, easy maintainability, and to support multi-threaded processing. The IP framework contains the lowest level image processing operators. The IPCC framework is the backbone of portable parallel processing in MIPS. The IPCC is completely portable to symmetric multi-processor platforms that use either Posix or Win32 thread libraries. The PIP framework contains the parallel processing versions of the IP framework and a set of processing functions that transparently implement, by using IPCC classes, parallel image processing. The IF framework contains the image format encoding and decoding classes. The VIE framework classes are the link between the user and the MIPS applications. VIE level objects enable runtime visual programming. They are represented as icons that are manipulated by the user to create arbitrarily complex processing chains.

2.2 MIPS Constructor

The second product is the MIPS Constructor. This portable, multi-threaded application is designed to allow the user to visually construct complex image processing algorithms by manipulating objects that are represented as icons. These image processing algorithms consist of reader/writer, data, processing, and display objects connected together in a processing chain. Processing chains automatically execute in parallel and can be of arbitrary complexity. All image data regardless of source (RGB/grayscale, video, or multi/hyperspectral images) is represented by a single data object – VieImage. VieImage is based on the idea of an image cube. The image processing objects in the Constructor all connect to VieImage objects. Hence, the image processing objects perform their processing functions regardless of the source of the image data. Thus, one can perform edge detection on an entire video stream or a grayscale image with the same processing chain. An overview of the Constructor interface is shown below in figure 2.

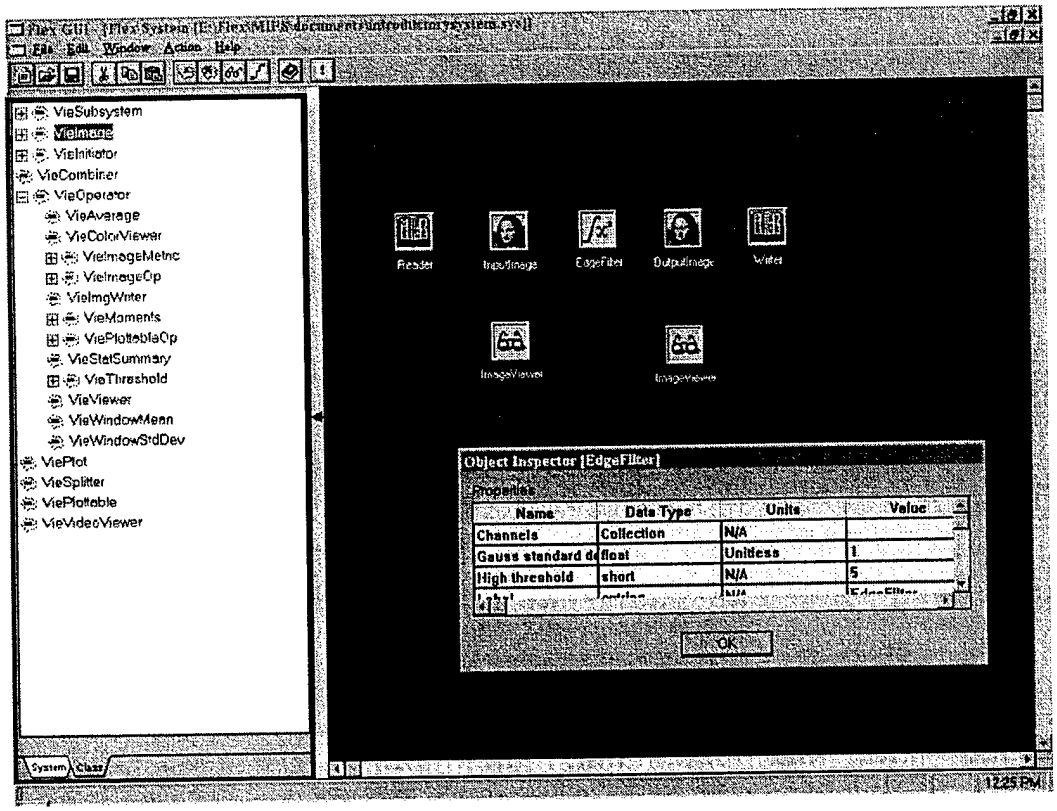


Figure 2. Overview of the MIPS Constructor interface.

The constructor will also display live plots and MPEG video streams. These fully portable features are implemented in Java and accessed vie the Java Native Interface. The MPEG viewer uses the Java Media Framework and native runtime codecs to display MPEG data. Figures 3 and 4 illustrate the live plotting facility and MPEG viewer.

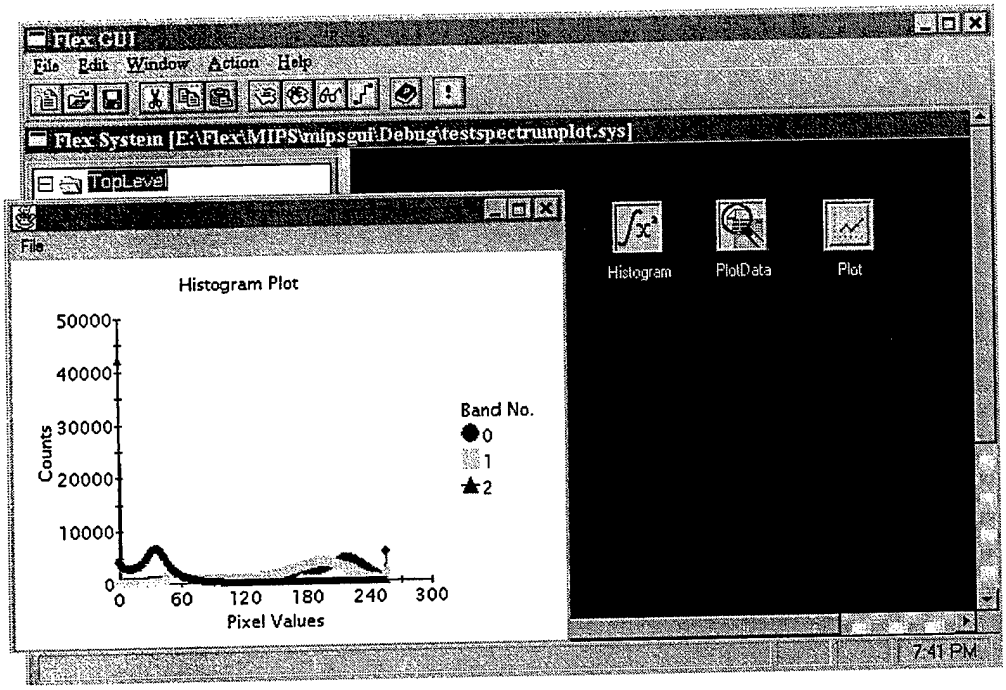


Figure 3. Java based live plotting implemented in MIPS.

Figure 4. Java based MPEG video viewer implemented in MIPS.

2.3 MIPS Mainwindow

The third product is the Mainwindow. The Mainwindow interface, shown in figure 5, is a window based image exploitation application, which has been integrated with the Constructor. The Mainwindow loads special archived processing chains (system files) at program initialization. Menus and submenus are created dynamically to correspond to each archived processing chain. Thus the Mainwindow can be field upgraded with new algorithms and capabilities by simply reading an additional set of system files. The internal processing is the same as that of the Constructor – parallel processing of all algorithmic operations.

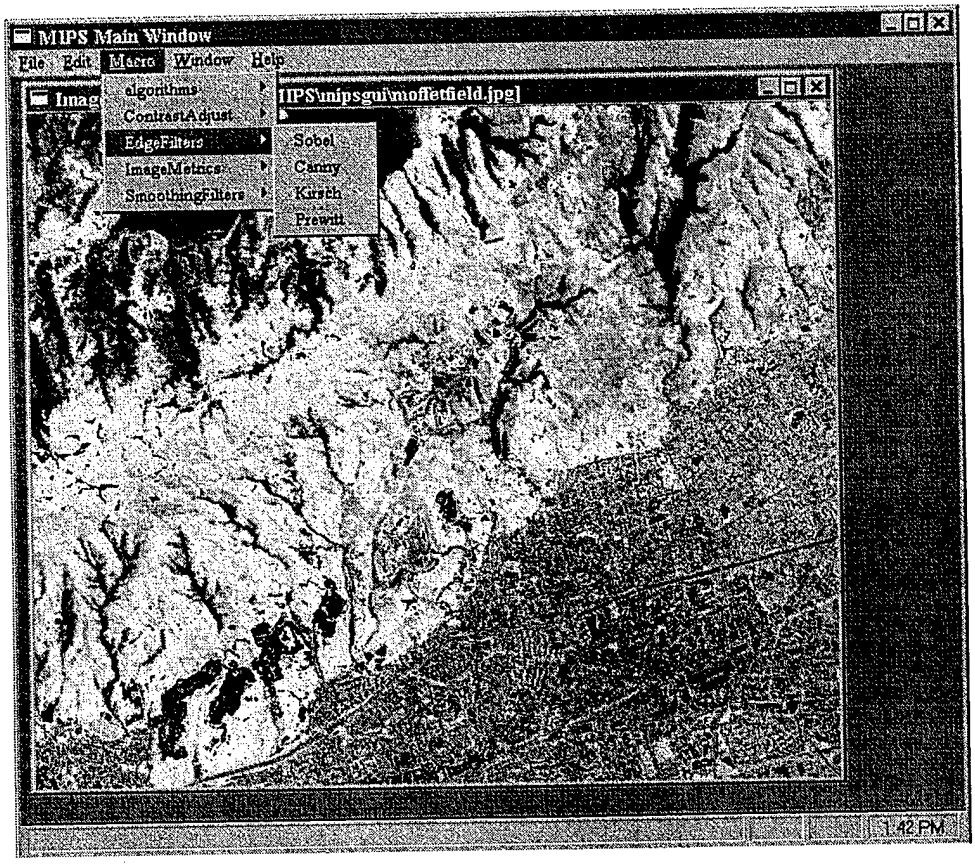


Figure 5. The Mainwindow interface showing the algorithm menu system.

3 The MIPS Frameworks

The MIPS software suite is comprised of 5 frameworks – the InterProcess Communication and Control (IPCC) framework, the Image Processing (IP) framework, the Parallel Image Processing (PIP) framework, the Image Formats (IF) framework, and the Visual Imaging Environment (VIE) framework. In this section we will cover each of these frameworks with an emphasis on how to create new MIPS Constructor and Mainwindow compatible VIE classes.

Many of the complexities of "data centric" parallel processing have been hidden in high level classes. Thus the would-be programmer can concentrate on writing portable, cross platform, parallel image/data processing algorithms without having detailed knowledge of the underlying mechanics, namely portable multi-threaded processing.

3.1 IPCC Framework

The classes in the IPCC framework are used to create and manage multiple threads and processes. The thread, process and mutex classes are built using the bridge pattern to assure link level portability across operating systems and platforms. The synchronization classes are written in terms of the thread interface class and hence are portable because the thread class is portable.

Thread is the abstract base class for all derived thread types. It provides a pure virtual function process, that when overridden by derived classes, will be executed when the child thread begins execution. This arrangement means that derived classes of Thread can define, for their own purposes, the structure and form of the work performed by overriding the method process.

The classes WorkerThread, WorkPile, Work, and ThreadSet are the workhorse of MIPS parallel processing. WorkerThread is a derived class of Thread. It is a template class with two template parameters - _OBJTYPE and _WORKTYPE. The policy enforced for _OBJTYPE is simply a doWork method that takes an object of type _WORKTYPE as a parameter. The policy enforced for _WORKTYPE consists of two functions, next() and end(), and a comparison operator >. The method next() should get the next available piece of work and the method end() should signal the end of the work in some fashion compatible with the comparison operator and an object of type _WORKTYPE. Because a WorkPile object is instantiated with an object of type _WORKTYPE in the WorkerThread constructor, objects of type _WORKTYPE have extra policy constraints imposed by the class WorkPile.

As was stated previously, WorkPile is a template class that takes a template parameter _WORKTYPE. WorkPile allows the thread system to synchronously process the current piece of work, decrement the amount of work in the "workpile", and test for an end condition. The next() method gets, sequentially, the next piece of work by returning the current value of on object of

type _WORKTYPE and decrement the member m_work. The end method returns an object of type _WORKTYPE constructed with a zero as an argument.

Work is an abstraction that describes how much work is yet to be processed. It is a base class that provides a basic functionality as a counter that tracks the current value of the number of "chunks" or pieces of work remaining. This class or its descendents are used with the class WorkPile. However, the concepts were kept separate so that the developer could redefine what is meant by work and still be compatible with the WorkerThread class.

ThreadSet is a set or group of threads with a method that implements the so-called barrier synchronization method. Derived classes of Thread may be added or deleted to the set. The method wait() waits for all threads in the set to finish execution before relinquishing control to the called method. Hence, wait() implements barrier synchronization in that all threads must join at the barrier before the caller can proceed.

These classes cooperate to allow multiple threads to automatically process data until no more work is left to perform. Figure 6 below illustrates this process in the context of image data.

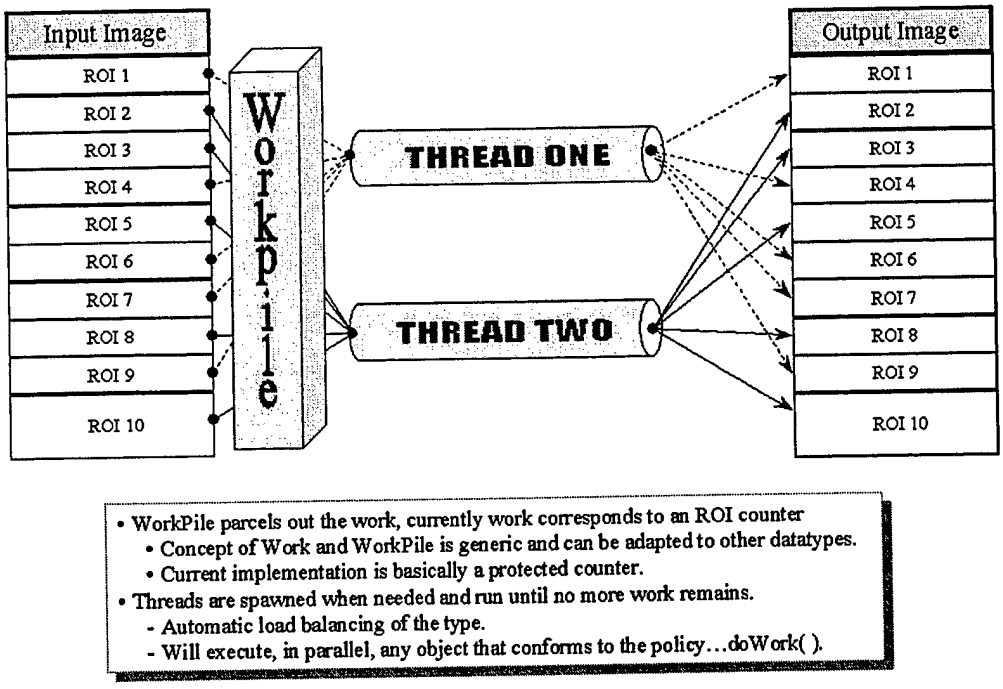


Figure 6. Graphical depiction of the MIPS "data centric" parallel processing.

Two template functions are provided that hide the details of using the WorkerThread, WorkPile, Work, and ThreadSet of classes. The first one is ApplyParallel. It is a general-purpose function

that takes two parameters – an object with an appropriately defined doWork method and a Work object that has been constructed with the number of chunks to process. A call to ApplyParallel will automatically setup the thread system, run the contents of doWork in parallel, and wait for the threads to terminate. The other template function is launch. Launch is a higher abstraction than ApplyParallel. It takes a single argument, the object with the doWork method, and automatically creates an appropriate Work object, calling ApplyParallel to start the parallel processing.

Table 1 below summarizes the IPCC classes.

Class Name	Description
ApplyParallel	Function that creates and executes a parallel processing job.
Guard	Provides mutex synchronization for a calling scope.
launch	Function that creates and executes a parallel processing job using ApplyParallel.
Mutex	Base interface class for mutex synchronization objects.
Process_	Base interface class for spawning and executing heavy weight processes.
Thread	Abstract base thread class. Defines much of the interface for working with threads.
ThreadSet	A container class for subclasses of Thread. Also contains a barrier synchronization method.
Work	A base class that defines the notion of work as a counter.
WorkPile	Manages how the Work is parceled out among the WorkerThreads.
WorkerThread	Derived class of Thread. WorkerThread processes data in pre-defined chunks.

3.2 IP Framework

The IP framework is a collection of classes that perform some image processing function on a region of interest (roi) of an image. These classes are unaware of any parallelism and can potentially be used in an other applications where parallelism is not required. The IP classes are the low level image processing operators of the MIPS software suite. The PIP framework uses these classes as a basis for developing parallel aware operators. Since these classes are all templates (meaning there are no .cpp files to compile) there are no build instructions associated with these classes.

There are several base classes within this framework that provide member variables, such as pointers to input and output images, to derived classes that relieve the developer of new classes from duplicating these member variables. These classes are ImageOperator and ImageOutputOP. ImageOperator provides to derived classes a pointer to an input image. All classes that require an input image but only create scalar output should derive from this class. Operator classes that require an input and output image pointer should derive from ImageOutputOP. All IP operators that are part of the MIPS software suite derive from one of these three classes. It should also be noted that MIPS operators do not overwrite input images with computed output values but rather writes the computed values to a new output data structure.

Table 2 below summarizes the IP framework classes:

Class Name	Description
AbsoluteOp	Computes the absolute value of pixels in a roi.
AccumulateOp	Accumulates pixel values from two inputs.
AddBitErrorOp	Adds bit error.
AddConstantOp	Adds a constant.
AddGuassOp	Adds gaussian noise.
AddImpulsiveOp	Adds impulsive noise.
AddOp	Adds coincident roi's from two images to form a third.
AddUniformOp	Adds uniform noise.
ApplyFunctionOp	Applies a function, contained in an input object, to the pixels in a roi.
ApplyMapOp	Applies an image map to pixels in a roi.
ATrimmedSqMeanOp	Applies the alpha trimmed square mean filter.
ClipValOp	Clips pixels in a roi between two values.
ConvolveAvgOp	Spatial convolution with a divisor applied after the kernel has been applied.
ConvolveOp	Spatial convolution of a kernel with pixels in a roi.
CopyOp	Copy's roi's from input to output.
CropImageOp	Crops the input image.
CrossGradientOp	Base cross gradient operator.
CrossGradientTOp	Thresholded version of the base class
CrossGradLevelBOp	Sets the background pixels to a constant.
CrossGradLevelGOp	Sets the edge pixels to a constant.
CrossGradLevelOp	Sets both the edge and background pixels to constants.
CrossMedianOp	Applies the cross median filter.
DifferentiateSpectrumOp	Differentiates the spectrum of each pixel (i,j) in a multi-channel image.
FlipHorzOp	Flips each roi horizontally.
FlipVertOp	Flips each roi vertically.
GtThresholdOp	Counts the number of pixel in a roi above a threshold.
HistogramOp	Computes the histogram of a roi.

HysteresisOp	Computes the hysteresis of a roi.
Image	The image cube.
ImageMap	Data structure representing image maps.
ImageOperator	Base class for operators needing only an input image pointer.
ImageOutputOP	Base class for operators needing both an input and output image pointer.
ImageROI	Class representing an image roi.
IntegrateSpectrumOp	Integrates the spectrum of a multi-channel image for each location (i,j)
IPExceptions	IP Framework exception classes.
Kernel	Data structure representing a convolution kernel.
LtThresholdOp	Counts the number of pixels in a roi below a threshold.
MaxPixelCompOp	Compares the pixels of two coincident roi's and outputs the largest one to the output roi.
MaxPixValOp	Determines the largest pixel value in a roi.
MedianOp	Applies a median filter.
MinPixValOp	Determines the smallest pixel value in a roi.
MIPSImage	The image cube instantiated with short, allows MIPS to process images of up to 16 bits of precision.
MIPSKernel	The kernel instantiated with short.
MultConstantOp	Multiplies a constant times each pixel in a roi.
NonmaxSuppressOp	Suppresses non local maximum pixels.
Norm2Op	Computes the norm between two coincident roi's.
Plottable	Class representing plottable data.
Rotate90Op	Rotates an roi by 90 degrees in the anti-clockwise direction.
ScaleOp	Zooms in or out of a roi.
SelectDynamicRangeOp	Extracts an 8 bit dynamic range from each pixel in a roi.
setBorder	Sets border pixels in a roi.
SetPixelsOp	Sets pixels in a roi to a value.
StatOp	Computes stats on a roi.
SubtractOp	Subtracts coincident roi's from two images to form a third.
SumOp	Computes the sum and sum of squares for pixels in a roi.

3.3 PIP Framework

The PIP framework is a collection of the parallel aware low level image processing operators and associated parallelizing template functions. Each operator class in the PIP framework contains a doWork method and is derived from an IP framework class. The template functions start the parallel processing for each operator type. A single call to one of these template functions will

perform, in parallel, the operation represented by the PIP operator. The parallel system will be started, the image data will be processed and written to output objects when one of these functions is called. For example, in the code fragment shown below, the absolute value of an input image will be computed (in parallel if possible) and written to the output image.

```
void someFunction( )
{
        MIPSImage inputImage,outputImage;

        // get the input image...in MIPS this is accomplished via references
        inputImage = getInputImage( );

        // now get the absolute value
        parallelAbsoluteVal(&inputImage,&outputImage);

        ...
}
```

The template function parallelAbsoluteVal is shown below:

```
template <class _IMAGETYPE>
void parallelAbsoluteVal( _IMAGETYPE* inputImage, _IMAGETYPE* outputImage )
{
        // Instantiate the parallel operator
        PAbsoluteOp<_IMAGETYPE> absoluteVal( inputImage, outputImage );
        // Launch the operator into the parallel system
        launch( absoluteVal );

}
```

The parallelizing template functions are a set of high level functions that give the developer access to a wide range of parallel operators. Developers can develop new template functions using the available PIP operators or they can develop template functions based on new PIP (and hence new IP) operators.

With respect to the image cube, these functions/classes process a single 2-dimensional channel at a time. To process an entire image cube in parallel, the template function must be embedded in a channel loop as shown below.

```
template <class _IMAGETYPE>
void someFunction(_IMAGETYPE* inImage, _IMAGETYPE* outImage, unsigned constant)
{
   ...
   // Since channel(i) returns a pointer each channel of the input image will be processed and written
   // to the associated channel in the output image
    for(unsigned i=0; i<inImage->channels(); i++)
      parallelAddConstant(inImage ->channel(i), outImage ->channel(i), constant);
   ...
}// end someFunction
```

Table 3 below summarizes the PIP framework classes:

Class Name	Base Class Functionality
PAbsoluteOp	Computes the absolute value of pixels in a roi.
PAccumulateOp	Accumulates pixel values from two inputs.
PAddBitErrorOp	Adds bit error.
PAddConstantOp	Adds a constant.
PAddGuassOp	Adds gaussian noise.
PAddImpulsiveOp	Adds impulsive noise.
PAddOp	Adds coincident roi's from two images to form a third.
PAddUniformOp	Adds uniform noise.
PApplyFunctionOp	Applies a function, contained in an input object, to the pixels in a roi.
PApplyMapOp	Applies an image map to pixels in a roi.
PATrimmedSqMeanOp	Applies the alpha trimmed square mean filter.
PClipValOp	Clips pixels in a roi between two values.
PConvolveAvgOp	Spatial convolution with a divisor applied after the kernel has been applied.
PConvolveOp	Spatial convolution of a kernel with pixels in a roi.
PCopyOp	Copy's roi's from input to output.
PCropImageOp	Crops the input image.
PCrossGradientOp	Base cross gradient operator.
PCrossGradientTOp	Thresholded version of the base class
PCrossGradLevelBOp	Sets the background pixels to a constant.
PCrossGradLevelGOp	Sets the edge pixels to a constant.
PCrossGradLevelOp	Sets both the edge and background pixels to constants.
PCrossMedianOp	Applies the cross median filter.
PDifferentiateSpectrumOp	Differentiates the spectrum of each pixel (i,j) in a multi-channel image.
PFlipHorzOp	Flips each roi horizontally.
PFlipVertOp	Flips each roi vertically.
PGtThresholdOp	Counts the number of pixel in a roi above a threshold.
PHistogramOp	Computes the histogram of a roi.
PHysteresisOp	Computes the hysteresis of a roi.
PIntegrateSpectrumOp	Integrates the spectrum of a multi-channel image for each location (i,j)
PLtThresholdOp	Counts the number of pixels in a roi below a threshold.

PMaxPixelCompOp	Compares the pixels of two coincident roi's and outputs the largest one to the output roi.
PMaxPixValOp	Determines the largest pixel value in a roi.
PMedianOp	Applies a median filter.
PMinPixValOp	Determines the smallest pixel value in a roi.
PMultConstantOp	Multiplies a constant times each pixel in a roi.
PNonmaxSuppressOp	Suppresses non local maximum pixels.
PNorm2Op	Computes the norm between two coincident roi's.
PRotate90Op	Rotates a roi by 90 degrees in the anti-clockwise direction.
PScaleOp	Zooms in or out of a roi.
PSelectDynamicRangeOp	Extracts an 8 bit dynamic range from each pixel in a roi.
PSetPixelsOp	Sets pixels in a roi to a value.
PStatOp	Computes stats on a roi.
PSubtractOp	Subtracts coincident roi's from two images to form a third.
PSumOp	Computes the sum and sum of squares for pixels in a roi.

Table 4 below summarizes the parallelized template functions of the PIP Framework.

Function Name	**Functionality**
parallelAbsoluteVal	Computes the absolute value of pixels in a roi.
parallelAccumulate	Accumulates pixels values from two inputs.
parallelAddBitError	Adds bit error.
parallelAddConstant	Adds a constant.
parallelAddGuass	Adds gaussian noise.
parallelAddImpulsive	Adds impulsive noise.
parallelAdd	Adds two images to form a third.
parallelAddUniform	Adds uniform noise.
parallelApplyFunction	Applies a function, contained in an input object, to the pixels in an image.
parallelApplyMap	Applies an image map to pixels in an image.
parallelATrimmedSqMean	Applies the alpha trimmed square mean filter.
parallelClipVal	Clips pixels in an image between two values.
parallelConvolveAvg	Spatial convolution with a divisor applied after the kernel has been applied.
parallelConvolve	Spatial convolution of a kernel with an image.
parallelCopy	Copy an image.
parallelCropImage	Crops the input image.
parallelCrossGradient	Cross gradient filter.
parallelCrossMedian	Applies the cross median filter.

parallelDifferentiateSpectrum	Differentiates the spectrum of each pixel (i,j) in a multi-channel image.
parallelFlipHorzontal	Flips the image horizontally.
parallelFlipVertical	Flips the image vertically.
parallelGtThreshold	Counts the number of pixels in an image above a threshold.
parallelHistogram	Computes the histogram of an image.
parallelHysteresis	Computes the hysteresis of an image.
parallelIntegrateSpectrum	Integrates the spectrum of a multi-channel image for each location (i,j)
parallelLtThreshold	Counts the number of pixels in an image below a threshold.
parallelMaxPixelComp	Compares the pixels of two input images and outputs the largest one to the output image.
parallelMaxPixVal	Determines the largest pixel value in an image.
parallelMedian	Applies a median filter.
parallelMinPixVal	Determines the smallest pixel value in an image.
parallelMultConstant	Multiplies a constant times each pixel in an image.
parallelNonmaxSuppress	Suppresses non local maximum pixels.
parallelNorm2	Computes the norm between two images.
parallelRotate90	Rotates an image by 90 degrees in the anti-clockwise direction.
parallelScale	Zooms in or out of an image.
parallelSelectDynamicRange	Extracts an 8 bit dynamic range from each pixel in an image.
parallelSetPixels	Sets pixels in an image to a value.
parallelStat	Computes stats on an image.
parallelSubtract	Subtracts two images to form a third.
parallelSum	Computes the sum and sum of squares for pixels in an image.

3.4 IF Framework

The IF framework is a cooperative set of classes that are used to encode and decode image, video, and multi/hyperspectral file formats. The interface for encoding and decoding is found in the abstract base class, ImageFormat. ImageFormat contains two methods, read() and write(), that must be overwritten by derived classes. The read() method contains the decoding instructions and the write() method contains the encoding instructions.

The MIPS Mainwindow and Constructor interfaces interact with the IF framework classes by extracting the proper format, based on a list of registered file extensions, from a metaclass pool.

This means, that on the gui side, there are no case or if statements to update or change when installing a new file format. File formats can be added or changed without changing any code outside of the IF framework. ImageFormat is the root class of the pool and all other formats are members of the pool. Several statements must be included in each derived format class so that they may be included in the format metaclass pool. In addition, a special constructor and a static method, extensions() must also be included in a new format class.

Table 5 below summarizes the IF framework classes.

Class Name	Class Description
AVIRISBRZFormat	AVIRIS Browse format (4 bands)
AVIRISFormat	AVIRIS format (all bands)
FileFilterCodex	Contains the file filter descriptions. (singleton)
FormatExtensionCodex	Contains descriptions of the legal extensions for all the format types. (singleton)
ImageFormat	Abstract base class for format types
ImageFormatMetaClass	Contains metaclass typedefs for the ImageFormat pool.
JPEGFormat	Encodes and decodes JPEG format
MPEGFormat	Encodes and decodes MPEG format
NITFFormat	Encodes and decodes NITF format
PPMRAWFormat	Encodes and decodes a simple binary PPM format
TIFFFormat	Encodes and decodes TIFF format
TFormatClass	Metaclass templates instantiated for ImageFormat

3.5 VIE Framework

Classes in the VIE framework are displayed and manipulated by the MIPS Constructor and Mainwindow. Parallel video, 2-dimensional, and multi/hyperspectral image processing algorithms are performed by classes in the VIE framework. Image input and output is controlled by classes in this framework as well as image display and plotting. These classes represent the high level of functionality in the MIPS software suite. Of primary concern to the developer are the methods for creating new VIE level classes and how to interact with the execution model that controls data processing.

Most VIE classes can be characterized by their input and output references. Generally, a class will have an input image reference and no output reference, an input image reference and an output image reference, an input image reference and an output plottable reference, or an input plottable reference and no output reference. Base classes exist for each of these cases to simplify development of new VIE classes. Table 6 below summarizes the set of VIE base classes.

Class Name	Reference Types
VieOperator	image input/ no output
VieImageOp	image input/ image output
ViePlottableOp	image input/ plottable output
ViePlot	plottable input/ no output

Programming new VIE level classes usually involves inheriting from one of the aforementioned base classes (to inherit the bulk of the required interface) and selectively overriding base class methods. Inheriting from one of the standard, supplied base classes will assure that the new class will be visible to the Constructor and Mainwindow gui. Taking advantage of the base classes through inheritance also means that the new classes will not have to "re-invent the wheel" with respect to references, attributes, metaclass and execution model related interface and functional requirements...most of the work has been done by the base classes. Overriding selective baseclass methods gives the new VIE class its individual behavior. The run method has not been implemented for any of the base classes, therefore any new derived VIE classes that processes data in some fashion during MIPS execution must override the run method. To create inputs unique to the newly created VIE class the getAttributes method must also be overridden. If new or different references are required then the getReferences method must be overridden. Other methods such as update or upkeep are occasionally overridden to define unique execution model behavior. VieAdd and VieSubtract override update due to special execution model requirements. Each must wait on two image objects to receive image data before these algorithm objects can proceed with their processing duties.

Table 7 below summarizes the VIE framework classes:

Class	Description
VieAboveThreshold	Count the number of pixels above a threshold
VieAdd	Add two images
VieAddBitError	Add bit error to an image
VieAddConstant	Add a constant to an image
VieAddGauss	Add gaussian noise to an image
VieAddImpulsive	Add impulsive noise to an image
VieAddNFrames	Add N consecutive frames of video
VieAddUniform	Add uniform noise to an image
VieATrimmedSqMean	Apply the alpha trimmed square mean filter
VieAverage	Compute the average pixel value

VieBelowThreshold	Count the number of pixels below a threshold
VieCannyEdge	Apply the Canny edge filter
VieCenterMean	Center the mean of the image
VieCombiner	Combine N images into an N channel image
VieContrastAdjust	Adjust the contrast
VieConvolve	Convolve a kernel with the image
VieCopy	Make a copy of the image
VieCropImage	Crop the image
VieCrossMedian	Apply the cross median filter
VieDifferentiateSpectrum	Differentiate the spectra at a wavelength
VieFlipHorizontal	Flip about the vertical axis
VieFlipVertical	Flip about the horizontal axis
VieFreiEdge	Apply the Frei Edge filter
VieGaussianSmooth	Apply a gaussian smoothing filter
VieHistogramEq	Equalize the histogram
VieImage	The image cube
VieImageHistogram	Compute the histogram
VieImageMax	Compute the max pixel value
VieImageMetric	Base class for VieImageMax and VieImageMin
VieImageMin	Compute the min pixel value
VieImageMoments	Compute the moments of an image
VieImageOp	Base class...
VieImgReader	Read an image from disc
VieImgWriter	Write an image to disc
VieInitiator	Base class...
VieIntegrateSpectrum	Integrate the spectrum of an image between two wavelengths
VieIntensityMap	Apply an intensity map to an image
VieInvertImage	Invert the image
VieKirschEdge	Apply the Kirsch edge filter
VieMacro	Mainwindow helper class
VieMoments	Moments base class
VieMpegEncoder	Encode frames to an MPEG file
VieMultConstant	Multiply a constant times an image
VieObj	Base class...
VieOperator	Base class...
ViePlot	Plot the contents of a plottable
ViePlottable	Data object that contains plottable data
ViePlottableOp	Base class...
ViePrewittEdge	Apply the Prewitt Edge filter
VieRobertsEdge	Apply the Roberts edge filter
VieRotate90Deg	Rotate the image by 90 degrees

VieScale	Zoom in or out
VieSelectDynamicRange	Select an 8 bit chunk of the dynamic range
VieServer	Base class...
VieSobelFilter	Apply the Sobel filter
VieSplitter	Split a multichannel image into N single channel images
VieSquareMedian	Apply the square median filter
VieStatSummary	Generate a statistics summary
VieStretchRange	Stretch the histogram
VieSubImageSpectra	Plot the spectra of a sub image
VieSubsystem	Base class...
VieSubtract	Subtract two images
VieSystem	Base class...
VieThreshold	Base class...
VieUnsharpMask	Apply an unsharp mask
VieVideoViewer	View the contents of an MPEG file
VieViewer	View up to 3 channels and 8 bits per channel of an image
VieWindowMean	Mean of a subimage
VieWindowStdDev	Standard deviation of a subimage

4 Constructor

The Constructor is used to create algorithms in the form of processing chains. These chains are composed of interacting objects, represented by icons, that cooperate to read, write, process and display image data. In order to understand how to create complex algorithms, one must understand how to assemble processing chains by placing objects on the object canvas, connecting the objects together and setting their parameters.

4.1 The Work Area: Object Canvas

The object canvas is the working space or "bread board" for connecting objects and setting their parameters. The user may change the position of object icons by selecting the icon and dragging it with the mouse. Any connections will "follow" the icon. Figure 7 illustrates the object canvas filled with objects and subsystems.

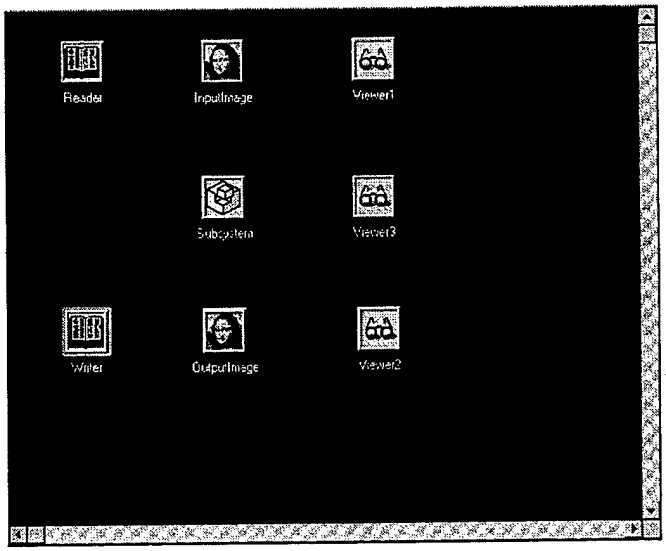


Figure 7. The object canvas containing a processing chain. Note the horizontal and vertical sliders used for viewing chains that are larger than the viewable size of the canvas.

4.2 Creating Objects: The Object Palette

The object palette, as shown in figure 2, has two components or views – the system view and the class view. Select a tab to show the corresponding view. The system view shows all the objects currently on the canvas. The class view shows all the I/O, data, processing, and display classes

that are available in the Constructor. The class view is also used to drag and drop objects, instances of classes, to the canvas.

4.2.1 System View

The system view shows all the objects in the current system. The tree of objects is organized with respect to containment in either the top level system or subsystems. A subsystem is denoted as a folder. If the folder is closed, , the subsystem is not currently selected. An open folder, , denotes a selected subsystem. Objects, represented as , are shown indented below their containing subsystem. Items on the same level are not indented with respect to each other. The top level system is a subsystem that contains all other subsystems and objects within a given system. The system view is illustrated in figure 8.

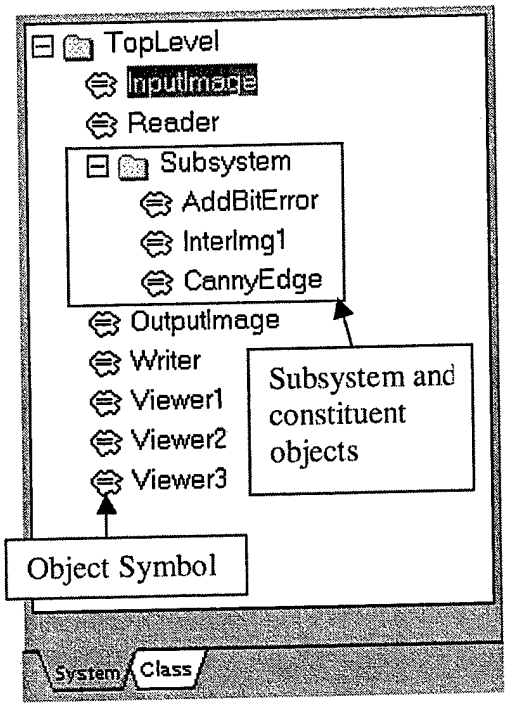


Figure 8. The system view of the object palette.

4.2.2 Class View

The class view contains the class tree. The class tree is a graphical depiction of the classes available in the Constructor. It is also the control from which objects are placed on the canvas. This is done by selecting a class in the class tree and "dragging and dropping" an object from the

tree to the canvas. The hierarchy of the tree is expressed in terms of inheritance, in the object-oriented sense. Descendent or derived classes are shown indented with respect to their ancestor or base classes. Some base classes are displayed to organize the class tree. Objects of these classes will not function in a processing chain. These display only classes are: VieInitiator, VieOperator, VieImageMetric, VieImageOp, VieMoments, ViePlottableOp and VieThreshold. Clicking on the plus sign, ⊞, will display nested classes. Clicking on the minus sign, ⊟, will collapse the branch showing only the base class. Figure 9 shows the class view of the object palette.

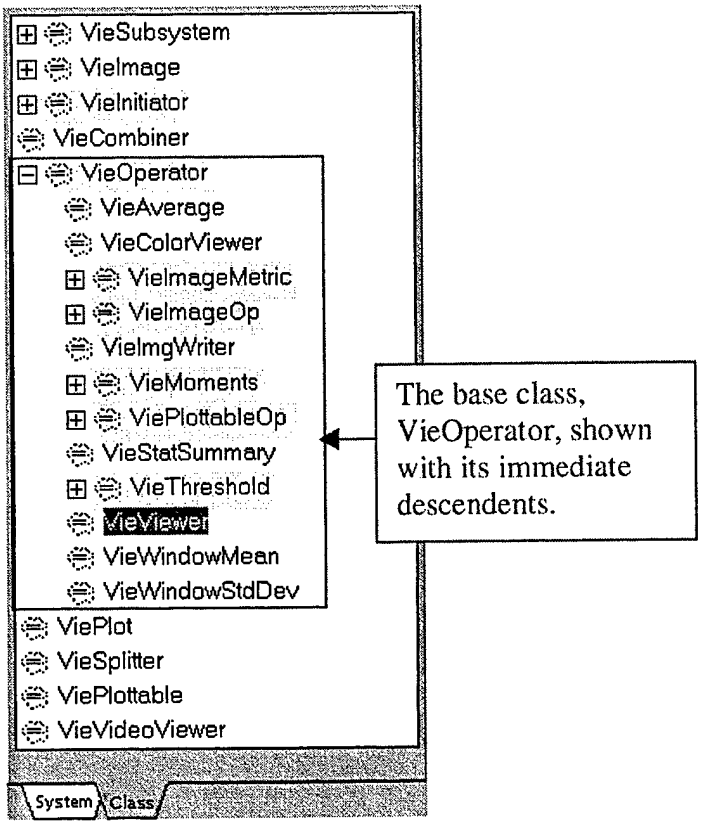


Figure 9. The class view of the object palette. Display only base classes are highlighted in red.

4.3 Object to Object Connections: References

A reference represents a connection between objects. A reference enables direct object-to-object communication and is represented, visually on the object canvas, by a red line segment connecting one object to another. The reference resolver is the tool that is used to graphically connect objects. Connecting objects via the reference resolver requires 5 steps.

Step 1. Select the object whose reference must be resolved. A connection is made ***from*** the object with the reference ***to*** the object that satisfies the reference.

Step 2. Press either the  button on the tool bar or invoke the Edit::References menu item. The reference resolver dialog will be displayed.

Step 3. Select the reference to resolve by selecting the appropriate row under *Reference Status* in the lower portion of the dialog. The row will be highlighted and a list of objects that can legally satisfy that particular reference will be listed under *Potential Resolvers* in the top portion of the dialog.

Step 4. From the list shown in the Potential Resolvers portion of the dialog, select the desired object with which to connect and press the *Connect button.* The name of the connected object will then appear under the *Value* column in the *Reference Status* portion of the dialog box.

Step 5. Press the *Ok button* to close the dialog box after all references have been resolved. Red lines will then appear between the connected objects.

To disconnect two objects follow the steps outlined above but instead of pressing Connect in step 4 press the *Disconnnect button.* At this point, the user may re-connect the disconnected reference to any object that appears in the *Potential Resolver* list. Figures 10 through 12 illustrate the use of the reference resolver.

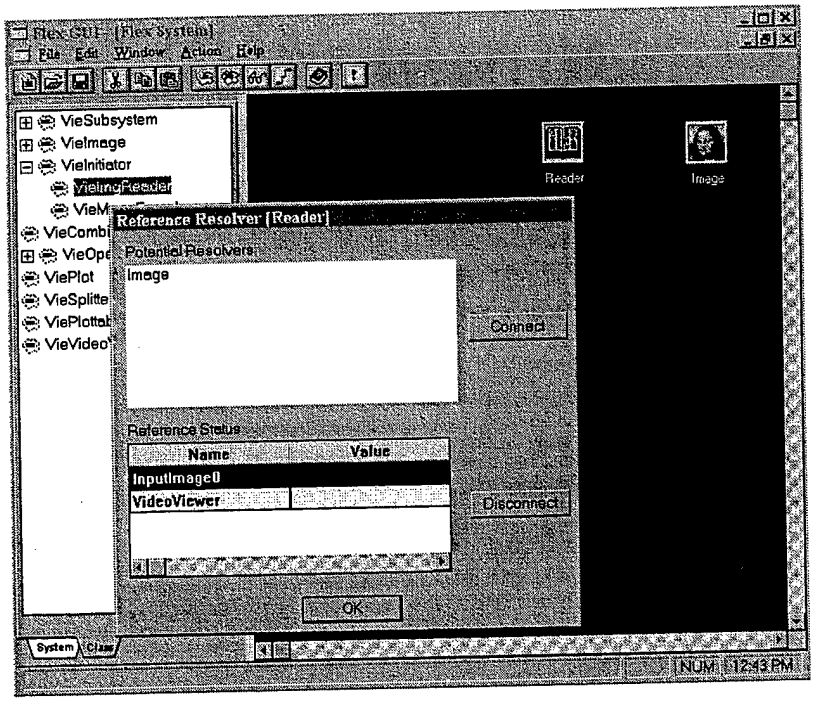


Figure 10. After invoking the reference resolver (steps 1 & 2) for the Reader object, the reference resolver dialog appears. Reader will be connected to Image. Note the red border signifying that the reader has at least one unsatisfied or unconnected reference.

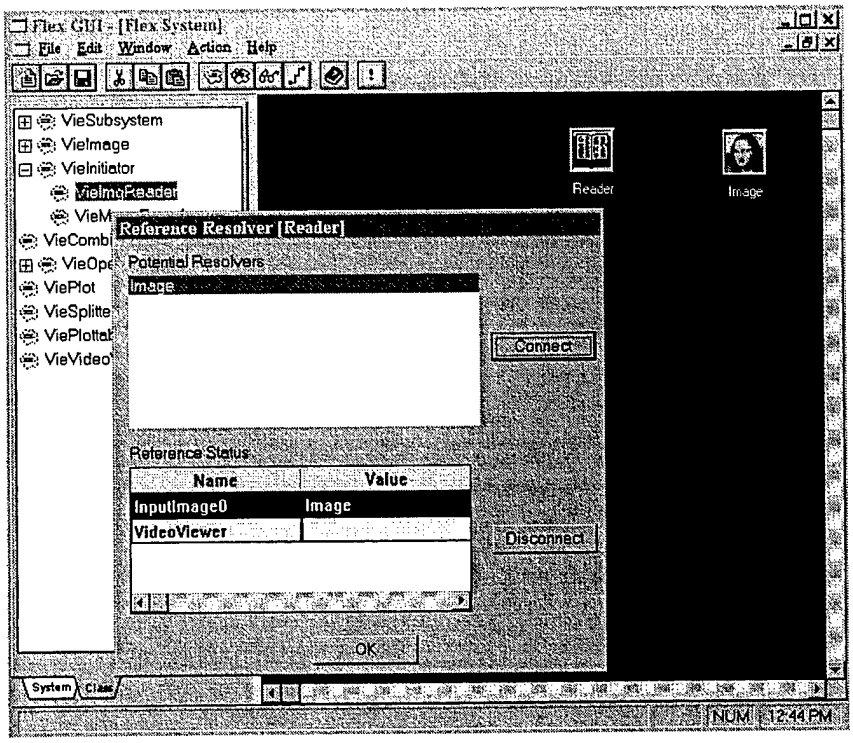


Figure 11. Appearance of the reference resolver after selecting the Image object from the list of potential resolvers and pressing the Connect button. Note the Image object is listed as the "Value" of the reference.

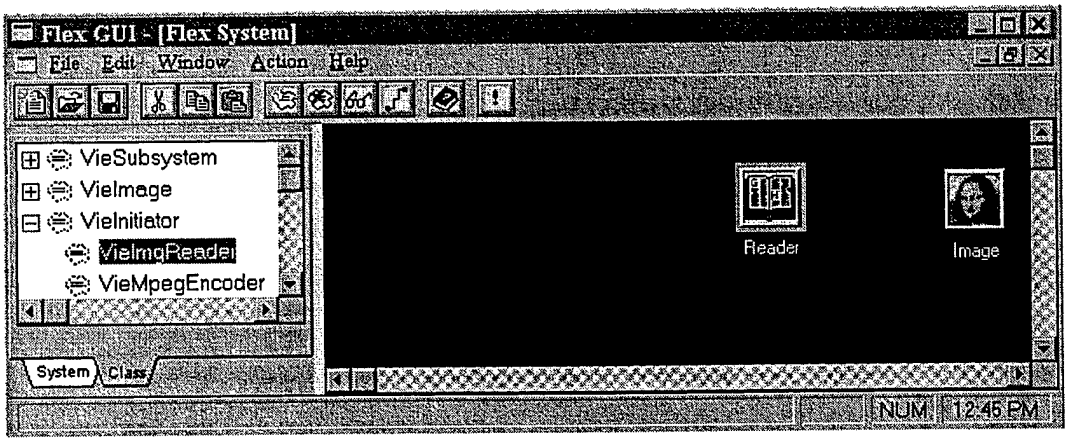


Figure 12. Once the Ok button is pressed the reference resolver dialog closes. The red line between Reader and Image appears after the dialog closes, signifying a reference connection from the reader object to the image object. Note the green border around Reader signifying that all required references have been resolved.

4.4 Object Parameters: The Object Inspector

In addition to references, Constructor objects also have parameters that control their behavior. Parameters are modified via a dialog called the Object Inspector. Parameters can be booleans, strings, filenames, integers, real numbers, and collections of any of these types. Modifying parameters involves 4 steps:

Step 1. Invoke the Object Inspector by double clicking on the icon representing the object whose parameters will be modified. In the case of subsystem objects the subsystem must be selected (a single click on the icon) and the Edit::Properties menu item invoked.

Step 2. In the object inspector, select the parameter to modify by placing the cursor on the appropriate row and click.

Step 3. If the parameter is not a collection but a single Boolean, string, filename, integer or real number an Edit box will appear. Place the cursor in the Value portion of the Edit box and enter the value. Press *OK* to close the Edit box.

If the parameter is a collection, another object inspector will appear. Choose the appropriate parameter in the collection (this may be another collection). Repeat, if necessary, until a single boolean, string, filename, integer or real number is encountered. At this point an edit box will appear and the user may enter the desired value. Edit the values in the collection, as required.

Step 4. Continue setting all the parameters as shown in steps 3 and 4 until all parameters have been set, then click *OK* in the original object inspector to exit the inspector and save the edited values.

Figures 13 – 16 illustrate the use of the object inspector.

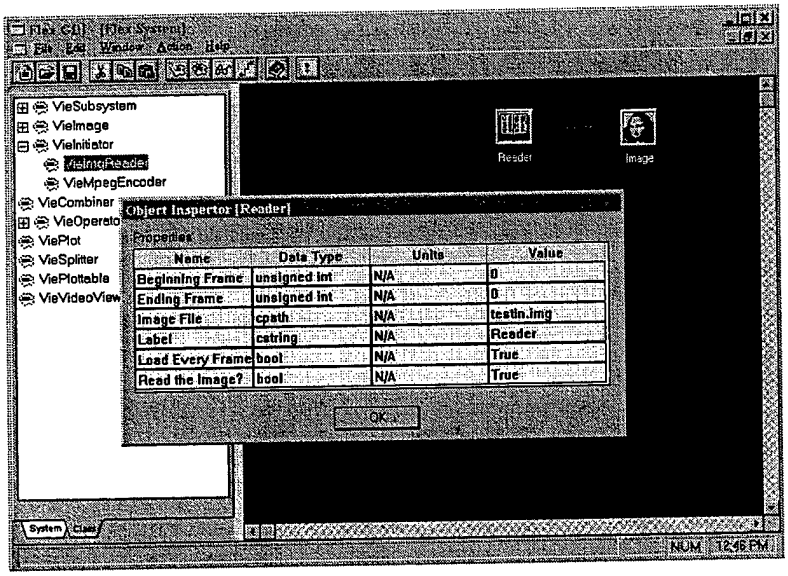


Figure 13. Illustrated above is the object inspector for the Reader object. Several types of parameters are available, each with their own editor.

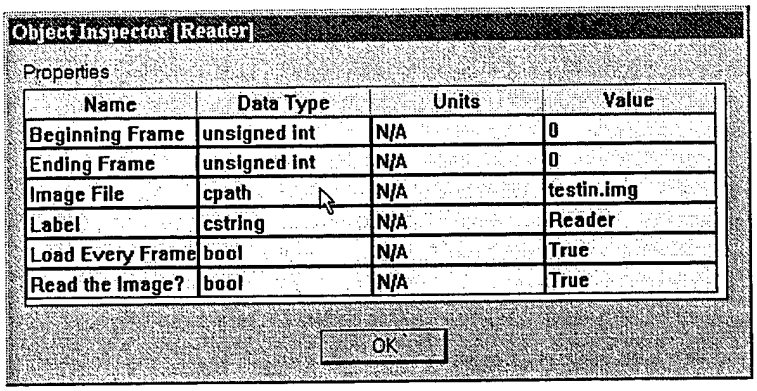


Figure 14. Select the parameter "Image File" to modify by placing the cursor on the appropriate row, clicking once with the left mouse button.

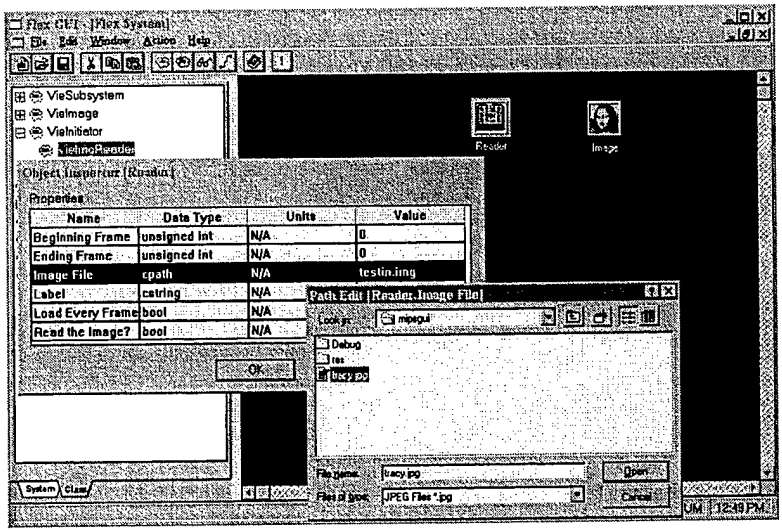


Figure 15. After selecting the "Image File" parameter, a file open dialog is presented. Find the desired file on disc and press *Open*.

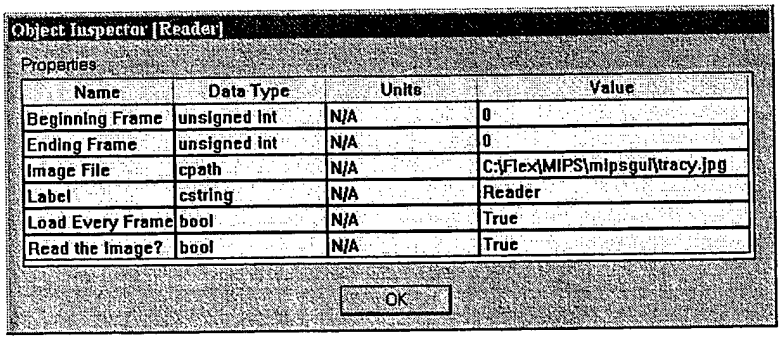


Figure 16. Result of the action described in figure 15.

Figures 17 through 20 illustrate the Boolean and Collection parameter editors.

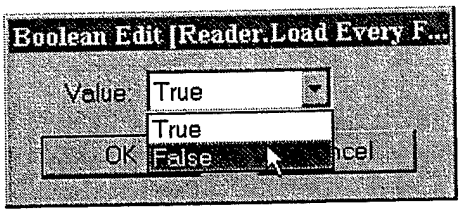


Figure 17. To set a value in the Boolean editor, select True or False from the pulldown list.

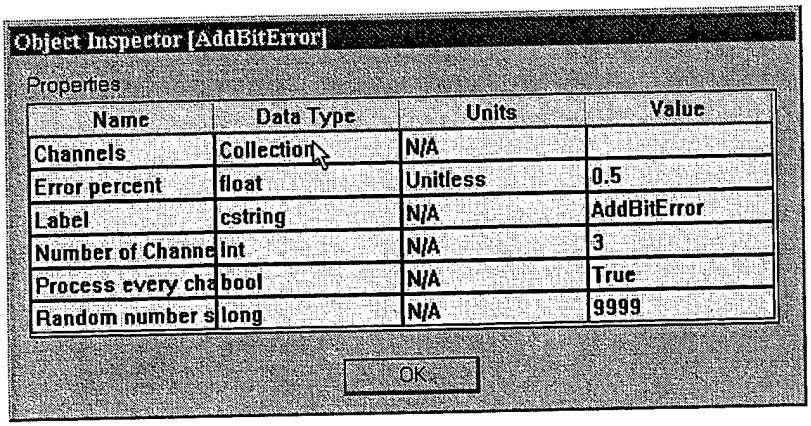


Figure 18. The parameter "Channels" is a collection. "Number of channels" is the parameter that controls the size of the collection. All collection input parameters have an accompanying size parameter. Operator objects automatically size collections that are used as output parameters.

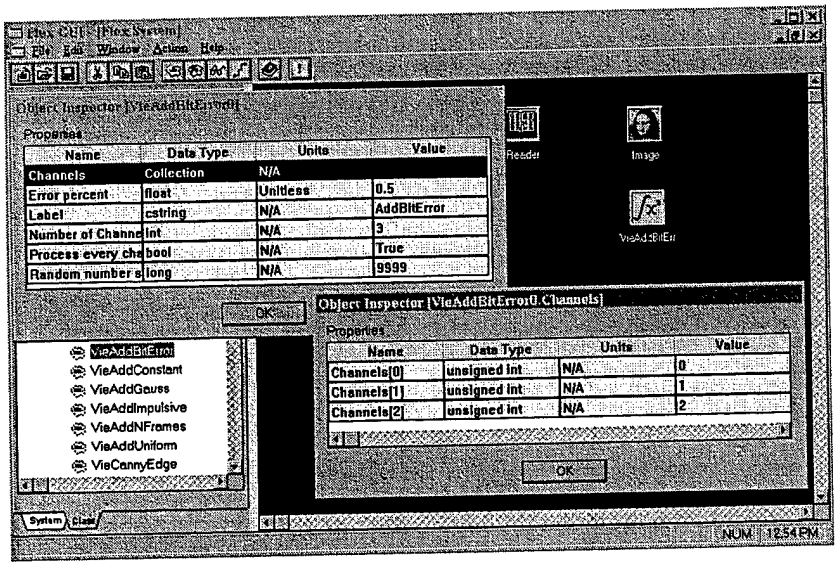


Figure 19. After setting the number of channels to 3, select the "Channel" parameter to display the collection. In this case the collection is a vector of channel or band numbers. Each of these can be edited independently.

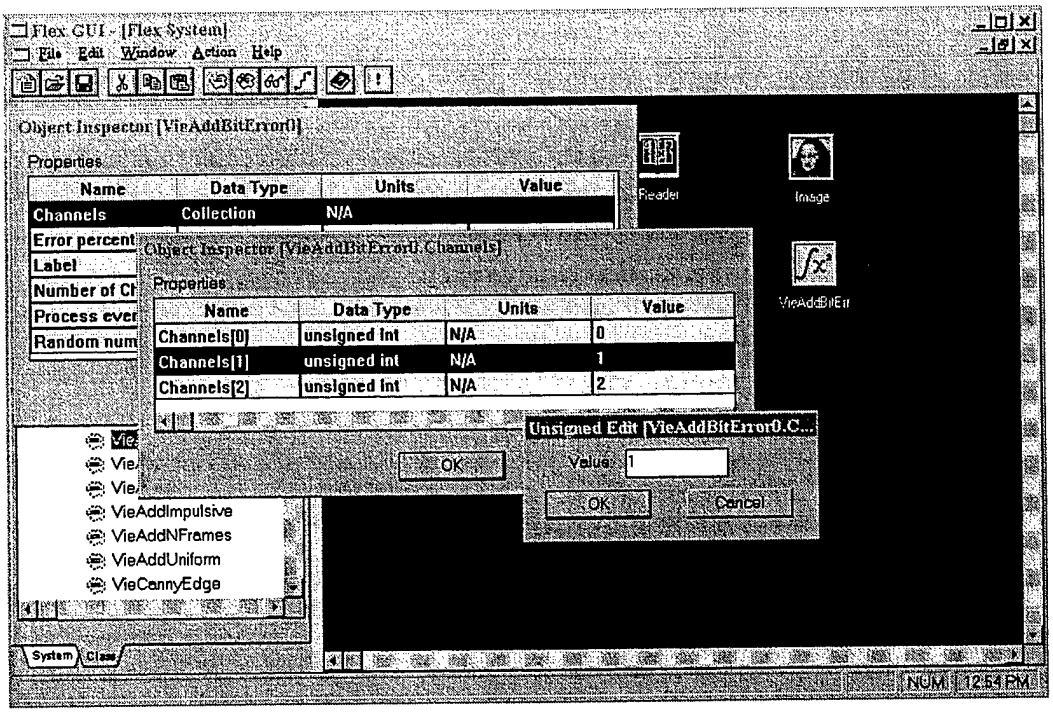


Figure20 Shown above is the appearance of the inspector while editing the value of the 2nd channel. Note channels are numbered from 0 to N-1, where N is the number of channels.

5 The Mainwindow

The Mainwindow, shown below in figure 21, is tightly integrated with the Constructor. It is remotely upgradeable by incorporating, at program initialization, Constructor produced processing chains, that have been archived into system files. It reads system files from a special directory and loads the algorithms from these system files into the Macro menu. A unique feature of the Mainwindow is that the macros are not interpreted or translated but run at the speed of native compiled code. Also the Mainwindow macro based algorithms retain all their parallel processing abilities.

The special macro directory is installed as ../Program/macros by the MIPS installer but can be changed to another directory by editing the file, ***mainwin.prf***, and changing the parameter "MACRO_DIRECTORY=./Somedir/anydir/". Note that the path must end with a slash. Standard directory navigation symbols apply such as "." and ".." .

Double clicking on the mainwin.exe icon launches the mainwindow. An open image must be present for the algorithms to take effect. The selected algorithm will be applied to the image window with focus. A single undo buffer allows for a one level undo. Algorithms can be repeatedly applied to the open image but the undo only reverts to the previously displayed image.

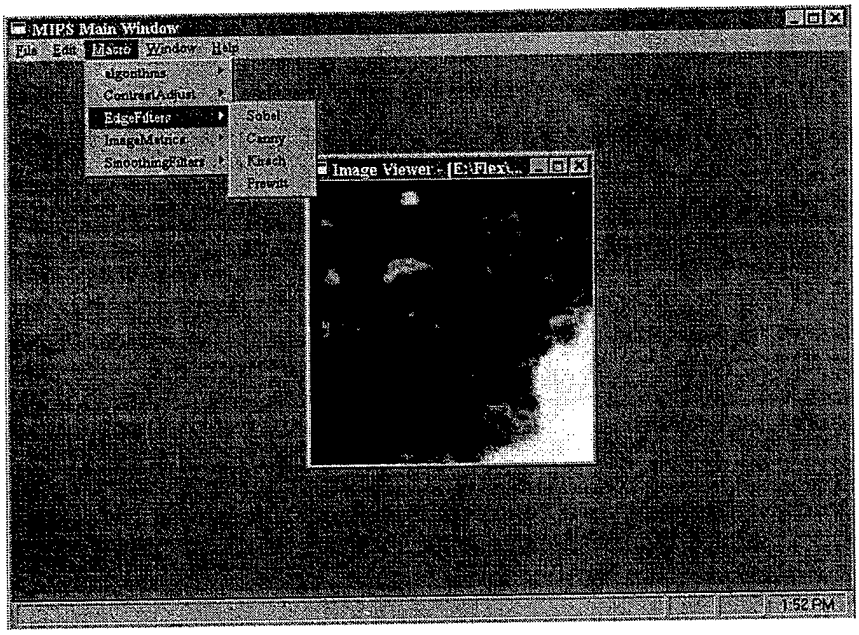


Figure 21. Mainwindow organization and appearance. Note that the system file EdgeFilters.sys contains VieMacros labeled Sobel, Canny, Kirsch, and Prewitt.

To apply an algorithm to an image, select the algorithm category then the specific algorithm. The inspector for the appropriate macro will be displayed, the user has the flexibility to change parameters at this point. After the Ok button has been pushed the algorithm will be applied to the image and the result will be displayed. The following figures illustrate this process.

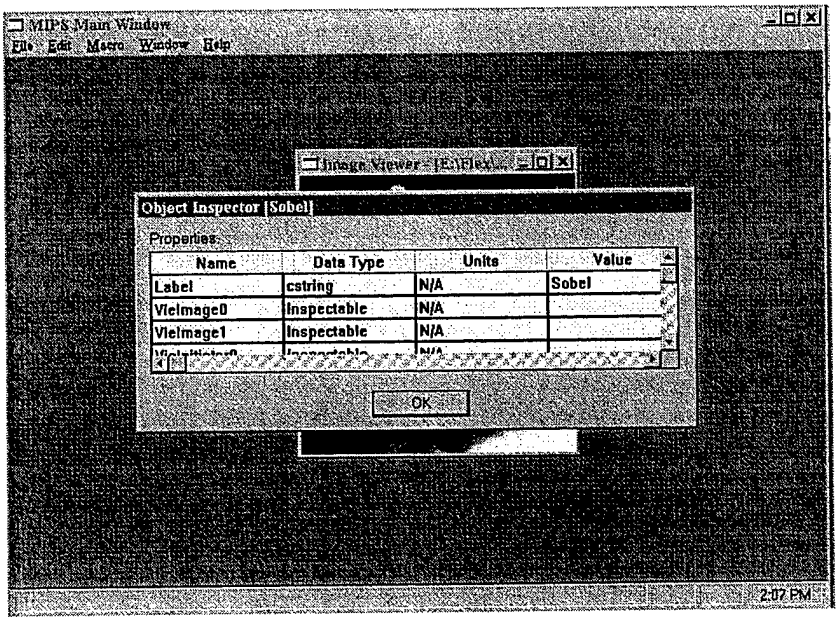


Figure 22. The inspector is displayed to further enhance flexibility.

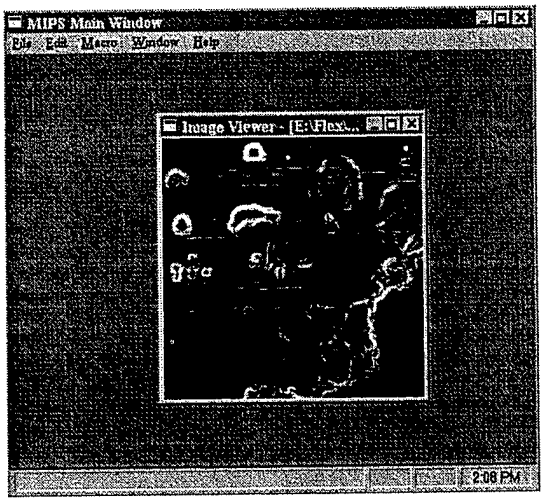


Figure 23. The result of the application of the Sobel Filter is displayed. Undo will restore the image to its previous state.

The basic method for creating a set of related algorithms for inclusion into the main window is to create a new system file, add the number of desired VieMacro objects to the top level, re-label the macro subsystems, then navigate into each macro subsystem and create the desired algorithm. Note, that if a new set of algorithms is created the main window must be restarted to load the new algorithms. The following figures outline creation of macros for inclusion in the main window.

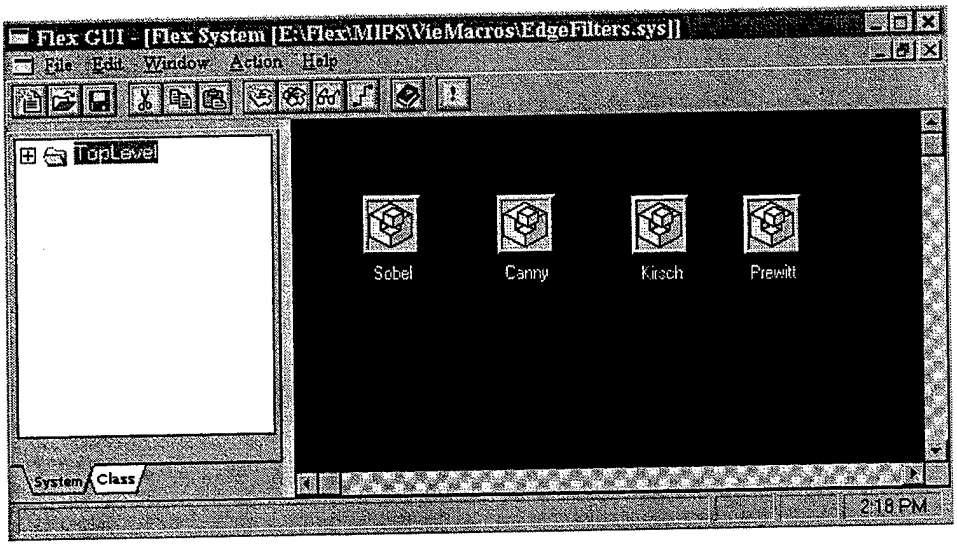


Figure 24. Sample system, EdgeFilters.sys, viewed at the top level.

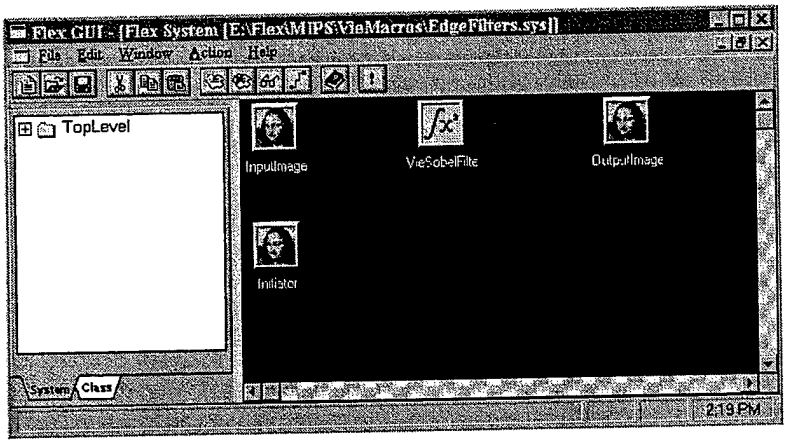


Figure 25. The Sobel filter macro contents.

The following is an example of a macro containing a more complicated algorithm.

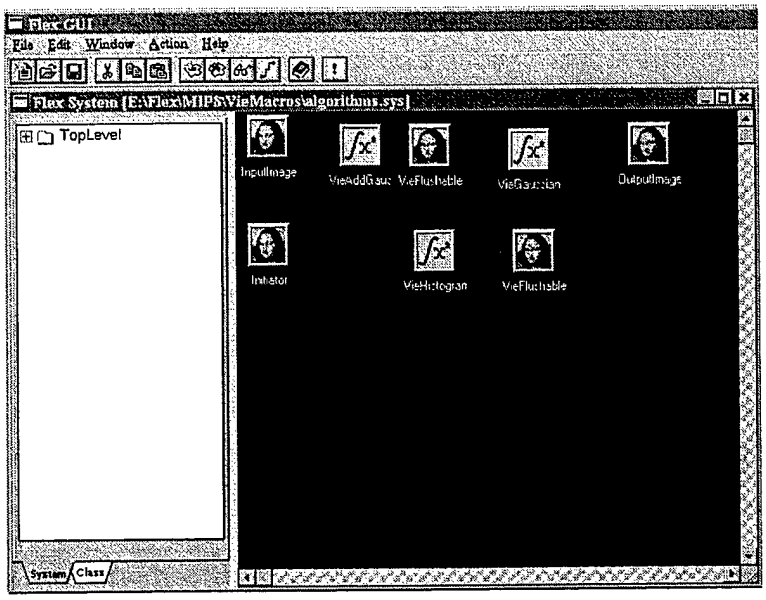


Figure 26. Complex macro algorithm. First gaussian noise is added to the image, then the contrast is stretched using histogram equalization, and finally the result is smoothed using a gaussian smoothing filter. Note the use of Flushable intermediate images.

6 Conclusion

In the context of today's fast paced electronic battlefield, improved information flow to the warfighter is a requirement. Fast, portable, and upgradeable image exploitation tools contribute to improving the flow of information. In response to this requirement NRC has developed the MIPS suite of products. The MIPS suite responds to the needs of software developers, image analysts, and warfighters. The MIPS frameworks allow for easy extension of basic processing capability. The MIPS Constructor provides a framework for visual programming that gives the analyst unique capability to visually program new algorithms without writing any source code. The MIPS Mainwindow provides, to the warfighter, a simple menu based image exploitation tool, which quickly processes image data from a variety of sources - 2D, video, and multi/hyperspectral images. The tight integration of the Constructor with the Mainwindow means that algorithms can be developed in a lab and quickly sent to fielded systems, where upgrades are installed without any modification to the fielded systems. The Mainwindow can be instantly upgraded by transmitting a file by any means necessary – email or removable storage media.

During the execution of the MIPS program NRC has:

- ...demonstrated the advantages of a multiprocessor workstation to exploit imagery,
- ...constructed a prototype workstation using cost effective commercial off-the-shelf components, integrating image and video analysis routines developed for the IE 2000,
- ...demonstrated the software reuse advantages of an object-oriented software development environment,
- ...and demonstrated an innovative, cost effective upgrade path for existing Air Force assets.

To execute the development of the MIPS software suite, NRC developed innovative solutions to the problems of platform/operating system portability, all source image exploitation, and real-time field upgradeable software.

The MIPS program has been successfully completed. NRC developed the MIPS frameworks and the MIPS applications – the Constructor and Mainwindow. The MIPS software (frameworks and applications) and the hardware platform were delivered to the AFRL/IFEC Image Exploitation 2000 (IE 2000) facility.

7 Recommendation for Future Activities

NRC's successful completion of the MIPS program has created the foundation for an end-to-end image exploitation system. The following recommendations are proposed.

- Implementation of a broader selection of algorithms to include material classification, scene segmentation, and frame-to-frame video processing.
- Enhance the existing user interface.
- Develop "hot swappable" domain object sets.
- Implement a hardware MPEG solution.
- Develop communication and network connectivity.

MISSION OF AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.